

S2**BTS Informatique de Gestion**
2nde année
Option **Administrateur de Réseaux Locaux d'Entreprise****Thème 4 : Le SQL****Dossier 02****Le langage de définition des données.****Objectifs**

Maîtriser les instructions de création, modification, suppression des bases de données, des tables et des contraintes d'intégrité référentielles.

Maîtriser la gestion des vues.

Plan de la séquence

1. Les tables.
 - 1.1. Création des tables.
 - 1.2. Les types de données.
 - 1.3. Création d'une table à partir d'une autre.
 - 1.4. Modification d'une table.
 - 1.5. Suppression d'une table
 - 1.6. Ajout de commentaires.
2. Les données.
 - 2.1. Insertion de données.
 - 2.2. La suppression des données.
3. Les contraintes d'intégrité.
 - 3.1. Définition d'une clé primaire.
 - 3.2. Définition d'une clé étrangère.
 - 3.3. Définition d'un index.
 - 3.4. Suppression d'une clé primaire, étrangère ou d'un index.
4. Les vues.
 - 4.1. Définition des vues.
 - 4.2. Création d'une vue.
 - 4.3. Suppression d'une vue.

TD/TP associé

TP01 Dossier 2 : LDD – Exercices de découverte.

Ressources

www.btsinfogap.org ; www.commentcamarche.net

Note importante : étant un dossier de rappel et/ou de révision, le contenu est partiellement issu du site www.commentcamarche.net. Vous pouvez vous y référer pour retrouver tout ceci.

1. Les tables.

1.1. Création des tables.

La création de tables se fait à l'aide du couple de mots-clés *CREATE TABLE*. La syntaxe de définition simplifiée d'une table est la suivante :

```
CREATE TABLE Nom_de_la_table (Nom_de_colonne1 Type_de_donnée,
                               Nom_de_colonne2 Type_de_donnée,
                               ...);
```

Le nom donné à la table doit généralement (sur la plupart des SGBD) commencer par une lettre, et le nombre de colonnes maximum par table est de 254.

L'option *NOT NULL*, placée immédiatement après le type de donnée permet de préciser au système que la saisie de ce champ est obligatoire.

1.2. Les types de données.

La norme SQL définit un certain nombre de types de données, présents dans tous les SGBDR. Toutefois, certains peuvent proposer d'autres types :

Nom	Description
bit [(n)]	Suite de bits de longueur fixe
bit varying [(n)], varbit	Suite de bits de longueur variable
boolean, bool	Booléen (Vrai/Faux)
smallint, int2	Entier signé de 2 octets
integer, int, int4	Entier signé de 4 octets
bigint, int8	Entier signé de 8 octets
serial, serial4	Entier de 4 octets à incrémentation automatique
bigserial, serial8	Entier de 8 octets à incrémentation automatique
character varying [(n)], varchar [(n)]	Suite de caractères de longueur variable
character [(n)], char [(n)]	Suite de caractères de longueur fixe
real, float4	Nombre à virgule flottante de simple précision
double precision, float8	Nombre à virgule flottante de double précision
interval [(p)]	Intervalle de temps
numeric [(p, s)], decimal [(p, s)]	Nombre exact de la précision indiquée
text	Chaîne de caractères de longueur variable
date	Date du calendrier (année, mois, jour)
time [(p)] [without time zone]	Heure du jour
time [(p)] with time zone, timetz	Heure du jour, avec fuseau horaire
timestamp [(p)] [without time zone]	Date et heure
timestamp [(p) with time zone], timestamptz	Date et heure, avec fuseau horaire

1.3. Création d'une table à partir d'une autre.

Il est possible de créer une table en insérant directement des lignes lors de la création. Les lignes à insérer peuvent être alors récupérées d'une table existante grâce au prédicat *AS SELECT*. La syntaxe d'une telle expression est la suivante :

```
CREATE TABLE Nom_de_la_table (Nom_de_colonne1 Type_de_donnée,
                               Nom_de_colonne2 Type_de_donnée,
                               ...)
AS SELECT Nom_du_champ1, Nom_du_champ2, ...
FROM Nom_de_la_table2
WHERE Projection;
```

1.4. Modification d'une table.

Par modification de la table, on entend le renommage des tables, l'ajout, la modification, le renommage et la suppression de colonnes

✓ **Renommage d'une table.**

Il peut parfois être intéressant de renommer une table, c'est la clause *RENAME* qui permet cette opération. La syntaxe de cette clause est :

```
ALTER TABLE Nom_de_la_table RENAME TO Nouveau_nom_de_la_table
```

✓ **Ajout d'une colonne.**

Associée à la clause *ADD*, la clause *ALTER* permet l'ajout de colonnes à une table. La syntaxe est la suivante :

```
ALTER TABLE Nom_de_la_table ADD Nom_de_la_colonne Type_de_donnees
```

✓ **Modification du type d'une colonne.**

Associée à la clause *MODIFY*, la clause *ALTER* permet la modification du type de données d'une colonne. La syntaxe est la suivante :

```
MySQL : ALTER TABLE Nom_de_la_table MODIFY Nom_de_la_colonne Type_de_donnees
```

PostgreSQL :

```
ALTER TABLE Nom_de_la_table ALTER COLUMN Nom_de_la_colonne TYPE Type_de_donnees
```

✓ **Renommage d'une colonne.**

Associée à la clause *RENAME*, la clause *ALTER* permet la modification du type de données d'une colonne. La syntaxe est la suivante :

```
ALTER TABLE Nom_de_la_table
    RENAME Nom_de_la_colonne TO Nouveau_nom_de_la_colonne
```

✓ **Suppression de colonnes.**

La clause *ALTER* permet la modification des colonnes d'une table. Associée à la clause *DROP COLUMN*, elle permet de supprimer des colonnes. La syntaxe est la suivante :

```
ALTER TABLE Nom_de_la_table DROP COLUMN Nom_de_la_colonne
```

Il faut noter que la suppression de colonnes n'est possible que dans le cas où :

- la colonne ne fait pas partie d'une vue,
- la colonne ne fait pas partie d'un index,
- la colonne n'est pas l'objet d'une contrainte d'intégrité.

1.5. Suppression d'une table

La suppression d'une table se fait avec la syntaxe suivante :

```
DROP TABLE Nom_de_la_table;
```

Il faut noter que la suppression d'une table n'est possible que dans le cas où :

- la table n'est pas attachée à une vue,
- aucun champ de la table n'est l'objet d'une contrainte d'intégrité.

Si des contraintes d'intégrité référentielles existent dans la base de données, il faut supprimer les tables dans le bon ordre, ou supprimer toutes les contraintes d'intégrité référentielle avant de supprimer les tables, ou effectuer une suppression en cascade.

Suppression d'une contrainte d'intégrité référentielle :

```
ALTER TABLE Nom_Table_Avec_Contrainte DROP CONSTRAINT Nom_contrainte ;
```

Suppression d'une table en cascade :

```
DROP TABLE Nom_Table_Cible_Contrainte CASCADE;
```

1.6. Ajout de commentaires.

Grâce à la clause *COMMENT*, il est possible de documenter les tables ou les colonnes en leur ajoutant des commentaires, c'est-à-dire une description :

- soit de la table elle-même

```
COMMENT ON TABLE Nom_de_la_table IS 'Commentaires';
```
- soit de certaines colonnes en particulier

```
COMMENT ON COLUMN Nom_de_la_table.Nom_de_la_colonne IS 'Commentaires';
```

2. Les données.

2.1. Insertion de données.

```
INSERT INTO Nom_de_la_table ( Nom_de_la_colonne1, Nom_de_la_colonne2, ...)  
VALUES (valeur_colonne_1, valeur_colonne2, ...)
```

La partie (*Nom_de_la_colonne1*, *Nom_de_la_colonne2*, ...) est facultative si tous les champs doivent être sélectionnés.

Par définition, un champ de type *serial* permet de générer automatiquement un identifiant unique. Il n'est donc pas nécessaire, et même déconseillé, de lui imposer une valeur.

2.2. La suppression des données.

```
DELETE from Nom_de_la_table;
```

Il est également possible de supprimer uniquement les données en conservant la structure de la table grâce à la clause *TRUNCATE*.

```
TRUNCATE TABLE Nom_de_la_table ;
```

3. Les contraintes d'intégrité.

3.1. Définition d'une clé primaire.

Grâce à SQL, il est possible de définir des clés, c'est-à-dire spécifier la (ou les) colonne(s) dont la connaissance permet de désigner précisément un et un seul tuple (une ligne).

L'ensemble des colonnes faisant partie de la table en cours et permettant de désigner de façon unique un tuple est appelé **clé primaire** et se définit grâce à la clause *PRIMARY KEY* suivie de la liste de colonnes, séparées par des virgules, entre parenthèses. Ces colonnes ne peuvent alors plus prendre la valeur *NULL* et doivent être telles que deux lignes ne puissent avoir simultanément la même combinaison de valeurs pour ces colonnes.

3.1.1. Au moment de la création de la table.

✓ Dans le cas d'une clé simple :

```
CREATE TABLE Nom_de_la_table (
    Nom_de_colonne1 Type_de_donnée PRIMARY KEY,
    Nom_de_colonne2 Type_de_donnée,
    ...
);
```

ou

```
CREATE TABLE Nom_de_la_table (
    Nom_de_colonne1 Type_de_donnée,
    Nom_de_colonne2 Type_de_donnée,
    ...,
    PRIMARY KEY ( Nom_de_colonne_clé_primaire )
);
```

✓ Dans le cas d'une clé composée :

```
CREATE TABLE Nom_de_la_table (
    Nom_de_colonne1 Type_de_donnée,
    Nom_de_colonne2 Type_de_donnée,
    ...,
    PRIMARY KEY ( Nom_de_premiere_colonne_clé_primaire,
    Nom_de_seconde_colonne_clé_primaire )
);
```

Dans tous ces cas, la contrainte de clé primaire est automatiquement nommée *Nom_de_la_table_pkey*

3.1.2. Après la création des tables.

L'intérêt de cette méthode est de pouvoir définir le nom de la contrainte de clé primaire. Cela peut permettre de la supprimer, sans supprimer la table.

Le nom de la contrainte est précisé par *constraint Nom_contrainte_clé_primaire*. Ceci est facultatif, et dans ce cas, la contrainte de clé primaire est automatiquement nommée *Nom_de_la_table_pkey*

✓ **Dans le cas d'une clé simple :**

```
ALTER TABLE Nom_de_la_table
  ADD CONSTRAINT Nom_contrainte_clé_primaire
    PRIMARY KEY ( Nom_de_colonne_clé_primaire );
```

✓ **Dans le cas d'une clé composée :**

```
ALTER TABLE Nom_de_la_table
  ADD CONSTRAINT Nom_contrainte_clé_primaire
    PRIMARY KEY ( Nom_de_premiere_colonne_clé_primaire,
                  Nom_de_seconde_colonne_clé_primaire );
```

3.2. Définition d'une clé étrangère.

Lorsqu'une liste de colonnes de la table en cours de définition permet de définir la clé primaire d'une table étrangère, on parle alors de **clé étrangère**, et on utilise la clause *FOREIGN KEY*.

3.2.1. Au moment de la création de la table.

```
CREATE TABLE Nom_de_la_table (
    Nom_de_colonne1 Type_de_donnée,
    Nom_de_colonne2 Type_de_donnée,
    ...,
    FOREIGN KEY ( Nom_de_colonne_clé_étrangère )
      REFERENCES Nom_de_la_table_cible ( Nom_de_colonne_cible )
);
```

Dans ce cas, la contrainte de clé étrangère est automatiquement nommée *Nom_de_la_table_Nom_de_la_colonne_colonne_fkey*

Cette méthode a un défaut : la table cible de la clé étrangère doit obligatoirement être créée préalablement. Or ceci n'est pas toujours possible.

3.2.2. Après la création des tables.

```
ALTER TABLE Nom_de_la_table
  ADD CONSTRAINT Nom_contrainte_clé_primaire
    FOREIGN KEY ( Nom_de_colonne_clé_étrangère );
```

Le nom de la contrainte est précisé par *constraint Nom_contrainte_clé_étrangère*. Ceci est facultatif, et dans ce cas, la contrainte de clé étrangère est automatiquement nommée *Nom_de_la_table_fkey*

Cette méthode est à privilégier pour garantir le succès sans soucis de la création des clés étrangères.

3.3. Définition d'un index.

Un index est un objet complémentaire (mais non indispensable) à la base de données permettant d'"indexer" certaines colonnes dans le but d'améliorer l'accès aux données par le SGBDR, au même titre qu'un index dans un livre ne vous est pas indispensable mais vous permet souvent d'économiser du temps lorsque vous recherchez une partie spécifique de ce dernier...

Toutefois, la création d'index utilise de l'espace mémoire dans la base de données, et, étant donné qu'il est mis à jour à chaque modification de la table à laquelle il est rattaché, il peut alourdir le temps de traitement du SGBDR lors de la saisie de données. Par conséquent, il faut que la création d'index soit justifiée et que les colonnes sur lesquelles il porte soient judicieusement choisies, de façon à minimiser les doublons. Ainsi certains SGBDR créent automatiquement un index lorsqu'une clé primaire est définie.

La création d'index en SQL se fait grâce à la clause *INDEX* précédée de la clause *CREATE*. Elle permet de définir un index désigné par son nom, portant sur certains champs d'une table. La syntaxe est la suivante :

```
CREATE [UNIQUE] INDEX Nom_de_l_index  
ON Nom_de_la_table (Nom_de_champ [ASC/DESC], ...);
```

L'option *UNIQUE* permet de définir la présence ou non de doublons pour les valeurs de la colonne. Les options *ASC/DESC* permettent de définir un ordre de classement des valeurs présentes dans la colonne

3.4. Suppression d'une clé primaire, étrangère ou d'un index

Pour supprimer une contrainte de clé primaire ou étrangère :

```
ALTER TABLE Nom_de_la_table DROP CONSTRAINT Nom_contrainte;
```

Pour supprimer un index :

```
DROP INDEX Nom_de_l_index;
```

4. Les vues.

4.1. Définition des vues.

Une vue est une table virtuelle, c'est-à-dire dont les données ne sont pas stockées dans une table de la base de données, et dans laquelle il est possible de rassembler des informations provenant de plusieurs tables. On parle de "vue" car il s'agit simplement d'une représentation des données dans le but d'une exploitation visuelle. Les données présentes dans une vue sont définies grâce à une clause *SELECT*.

L'intérêt des vues est double :

- Gérer les droits d'accès sur les données ou les colonnes et non sur les tables,
- Rassembler en une seule "table" des données issues de plusieurs.

Il est important de savoir que le lien entre les vues et les tables associées est dynamique, à savoir que toute modification sur l'une des tables est répercutée sur la vue.

4.2. Création d'une vue.

La création d'une vue se fait grâce à la clause *CREATE VIEW* suivie du nom que l'on donne à la vue, puis du nom des colonnes dont on désire agrémente cette vue (il faut autant de redéfinitions de colonnes qu'il y en aura en sortie), puis enfin d'une clause *AS* précédant la sélection. La syntaxe d'une vue ressemble donc à ceci :

```
CREATE OR REPLACE VIEW Nom_de_la_Vue (colonnes)  
AS SELECT ...
```

OR REPLACE n'est pas obligatoire. Cela permet de remplacer la vue si elle existe déjà.

Il n'est pas nécessaire de préciser (*colonnes*). Dans ce cas, les colonnes de la vue héritent du nom des colonnes projetées par la requête de sélection qui a servi au remplissage.

Si (*colonnes*) est précisé, il faut qu'il ait le même nombre de colonnes que la requête de sélection qui a servi au remplissage.

Les vues ainsi créées peuvent être l'objet de nouvelles requêtes en précisant le nom de la vue au lieu d'un nom de table dans un ordre *SELECT*...

4.3. Suppression d'une vue.

La suppression d'une vue se fait avec la syntaxe suivante :

```
DROP VIEW Nom_de_la_vue;
```